下载APP

06 | 事件触发：各类 eBPF 程序的触发机制及其应用场景

2022-01-28 倪朋飞

《eBPF 核心技术与实战》 课程介绍 >

讲述：山荣

时长 12:06 大小 11.09M

你好，我是倪朋飞。

上一讲，我带你一起梳理了 eBPF 程序跟内核交互的基本方法。一个完整的 eBPF 程序通常包含用户态和内核态两部分：用户态程序通过 BPF 系统调用，完成 eBPF 程序的加载、事件挂载以及映射创建和更新，而内核态中的 eBPF 程序则需要通过 BPF 辅助函数完成所需的任务。

在上一讲中我也提到，并不是所有的辅助函数都可以在 eBPF 程序中随意使用，不同类型的 eBPF 程序所支持的辅助函数是不同的。那么，eBPF 程序都有哪些类型，而不同类型的 eBPF 程序又有哪些独特的应用场景呢？今天，我就带你一起来看看。

eBPF 程序可以分成几类？

eBPF 程序类型决定了一个 eBPF 程序可以挂载的事件类型和事件参数，这也就意味着，内核中不同事件会触发不同类型的 eBPF 程序。

根据内核头文件 include/uapi/linux/bpf.h 中 bpf_prog_type 的定义，Linux 内核 v5.13 已经支持 30 种不同类型的 eBPF 程序（注意， BPF_PROG_TYPE_UNSPEC表示未定义）：

复制代码

```
enum bpf_prog_type {
  BPF_PROG_TYPE_UNSPEC, /* Reserve 0 as invalid program type */
  BPF_PROG_TYPE_SOCKET_FILTER,
  BPF_PROG_TYPE_KPROBE,
  BPF_PROG_TYPE_SCHED_CLS,
  BPF_PROG_TYPE_SCHED_ACT,
  BPF_PROG_TYPE_TRACEPOINT,
  BPF_PROG_TYPE_XDP,
  BPF_PROG_TYPE_PERF_EVENT,
  BPF_PROG_TYPE_CGROUP_SKB,
  BPF_PROG_TYPE_CGROUP_SOCK,
  BPF_PROG_TYPE_LWT_IN,
  BPF_PROG_TYPE_LWT_OUT,
  BPF_PROG_TYPE_LWT_XMIT,
  BPF_PROG_TYPE_SOCK_OPS,
  BPF_PROG_TYPE_SK_SKB,
  BPF_PROG_TYPE_CGROUP_DEVICE,
  BPF_PROG_TYPE_SK_MSG,
  BPF_PROG_TYPE_RAW_TRACEPOINT,
  BPF_PROG_TYPE_CGROUP_SOCK_ADDR,
  BPF_PROG_TYPE_LWT_SEG6LOCAL,
  BPF_PROG_TYPE_LIRC_MODE2,
  BPF_PROG_TYPE_SK_REUSEPORT,
  BPF_PROG_TYPE_FLOW_DISSECTOR,
  BPF_PROG_TYPE_CGROUP_SYSCTL,
  BPF_PROG_TYPE_RAW_TRACEPOINT_WRITABLE,
  BPF_PROG_TYPE_CGROUP_SOCKOPT,
  BPF_PROG_TYPE_TRACING,
  BPF_PROG_TYPE_STRUCT_OPS,
  BPF_PROG_TYPE_EXT,
  BPF_PROG_TYPE_LSM,
  BPF_PROG_TYPE_SK_LOOKUP,
};
```

对于具体的内核来说，因为不同内核的版本和编译配置选项不同，一个内核并不会支持所有的程序类型。你可以在命令行中执行下面的命令，来查询当前系统支持的程序类型：

复制代码

```
bpftool feature probe | grep program_type
```

执行后，你会得到如下的输出：

复制代码

```
eBPF program_type socket_filter is available
eBPF program_type kprobe is available
eBPF program_type sched_cls is available
eBPF program_type sched_act is available
eBPF program_type tracepoint is available
eBPF program_type xdp is available
eBPF program_type perf_event is available
...
eBPF program_type ext is NOT available
eBPF program_type lsm is NOT available
eBPF program_type sk_lookup is available
```

在这些输出中，你可以看到当前内核支持 kprobe、xdp、perf_event 等程序类型，而不支持 ext、lsm 等程序类型。

根据具体功能和应用场景的不同，这些程序类型大致可以划分为三类：

第一类是跟踪，即从内核和程序的运行状态中提取跟踪信息，来了解当前系统正在发生什么。

第二类是网络，即对网络数据包进行过滤和处理，以便了解和控制网络数据包的收发过程。

第三类是除跟踪和网络之外的其他类型，包括安全控制、BPF 扩展等等。

接下来，我就带你一起分别看看，每一类 eBPF 程序都有哪些具体的类型，以及这些不同类型的程序都是由哪些事件触发执行的。

跟踪类 eBPF 程序

先看第一类，也就是跟踪类 eBPF 程序。

跟踪类 eBPF 程序主要用于从系统中提取跟踪信息，进而为监控、排错、性能优化等提供数据支撑。比如，我们前几讲中的 Hello World 示例就是一个 BPF_PROG_TYPE_KPROBE 类型的跟踪程序，它的目的是跟踪内核函数是否被某个进程调用了。

为了方便你查询，我把常见的跟踪类 BPF 程序的主要功能以及使用限制整理成了一个表格，你可以在需要时参考。

程序类型	功能描述	功能限制
BPF_PROG_TYPE_KPROBE	用于对特定函数进行动态插桩，根据函数位置的不同，又可以分为内核态 krpobe 和用户态 uprobe	内核函数和用户函数的定义属于不稳定 API，在不同内核版本中使用时，可能需要调整 eBPF 代码实现
BPF_PROG_TYPE_TRACEPOINT	用于内核静态跟踪点（可以使用 perf list 命令，查询所有的跟踪点）	虽然跟踪点可以保持稳定性，但不如 KPROBE 类型灵活，无法按需增加新的跟踪点
BPF_PROG_TYPE_PERF_EVENT	用于性能事件（perf_events）跟踪，包括内核调用、定时器、硬件等各类性能数据	需配合 BPF_MAP_TYPE_PERF_EVENT_ARRAY 或 BPF_MAP_TYPE_RINGBUF 类型的映射使用
BPF_PROG_TYPE_RAW_TRACEPOINT BPF_PROG_TYPE_RAW_TRACEPOINT_WRITABLE	用于原始跟踪点	不处理参数
BPF_PROG_TYPE_TRACING	用于开启 BTF 的跟踪点	需要开启 BTF

极客时间

这其中，KPROBE、TRACEPOINT 以及 PERF_EVENT 都是最常用的 eBPF 程序类型，大量应用于监控跟踪、性能优化以及调试排错等场景中。我们前几讲中提到的 BCC工具集，其中包含的绝大部分工具也都属于这个类型。

网络类 eBPF 程序

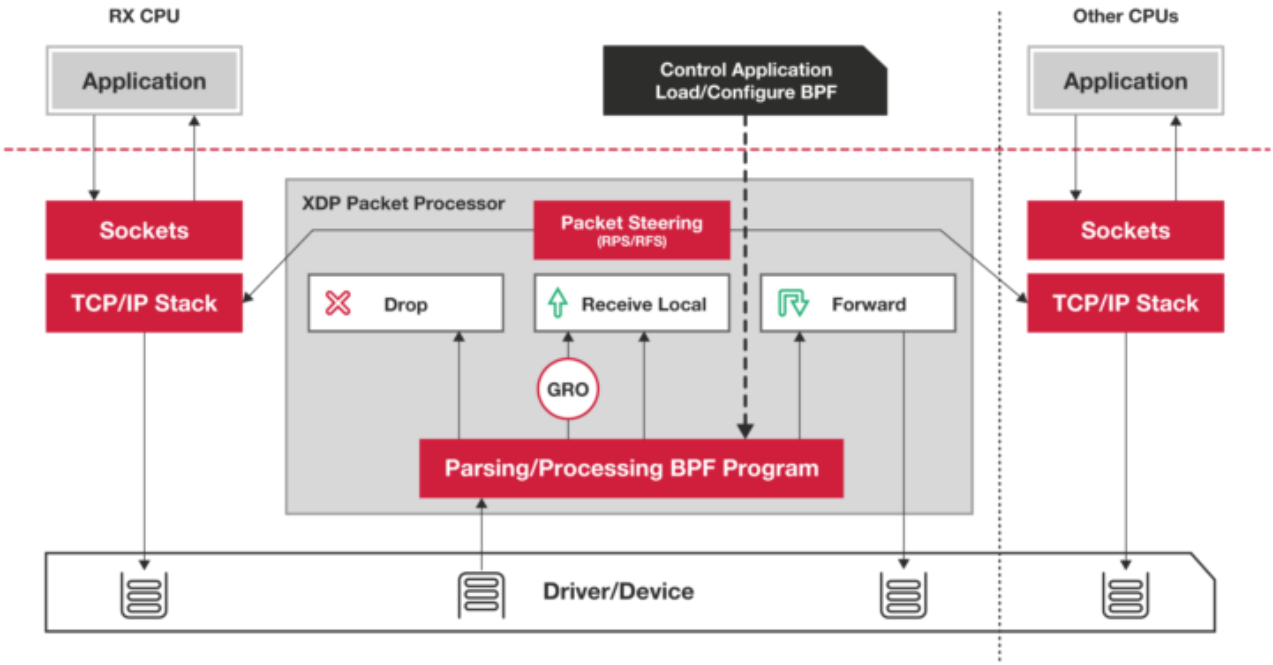
看完跟踪类 eBPF 程序，我们再来看看网络类 eBPF 程序。

网络类 eBPF 程序主要用于对网络数据包进行过滤和处理，进而实现网络的观测、过滤、流量控制以及性能优化等各种丰富的功能。根据事件触发位置的不同，网络类 eBPF 程序又可以分为 XDP（eXpress Data Path，高速数据路径）程序、TC（Traffic Control，流量控制）程序、套接字程序以及 cgroup 程序，下面我们来分别看看。

XDP 程序

XDP 程序的类型定义为 BPF_PROG_TYPE_XDP，它在**网络驱动程序刚刚收到数据包时**触发执行。由于无需通过繁杂的内核网络协议栈，XDP 程序可用来实现高性能的网络处理方案，常用于 DDoS 防御、防火墙、4 层负载均衡等场景。

你需要注意，XDP 程序并不是绕过了内核协议栈，它只是在内核协议栈之前处理数据包，而处理过的数据包还可以正常通过内核协议栈继续处理。你可以通过下面的图片（图片来自 iovisor.org）加深对 XDP 相对内核协议栈位置的理解：



XDP概览

根据网卡和网卡驱动是否原生支持 XDP 程序，XDP 运行模式可以分为下面这三种：

通用模式。它不需要网卡和网卡驱动的支持，XDP 程序像常规的网络协议栈一样运行在内核中，性能相对较差，一般用于测试；

原生模式。它需要网卡驱动程序的支持，XDP 程序在网卡驱动程序的早期路径运行；

卸载模式。它需要网卡固件支持 XDP 卸载，XDP 程序直接运行在网卡上，而不再需要消耗主机的 CPU 资源，具有最好的性能。

无论哪种模式，XDP 程序在处理过网络包之后，都需要根据 eBPF 程序执行结果，决定数据包的去处。这些执行结果对应以下 5 种 XDP 程序结果码：

结果码	含义	使用场景
XDP_DROP	丢包	数据包尽早丢弃可以减少 CPU 处理时间，因而常用于防火墙、DDoS 防御等丢弃非法包的场景
XDP_PASS	传递到内核协议栈	内核协议栈接收到网络包，按正常流程继续处理
XDP_TX XDP_REDIRECT	转发数据包到同一网卡/不同网卡	数据包在 XDP 程序修改后转发到网卡中，继续按正常的内核协议栈流程处理，常用在负载均衡中
XDP_ABORTED	错误	XDP 程序运行错误，数据包丢弃并记录错误行为，以便排错

极客时间

通常来说，XDP 程序通过 `ip link` 命令加载到具体的网卡上，加载格式为：

复制代码

```
# eth1 为网卡名
# xdpgeneric 设置运行模式为通用模式
# xdp-example.o 为编译后的 XDP 字节码
sudo ip link set dev eth1 xdpgeneric object xdp-example.o
```

而卸载 XDP 程序也是通过 `ip link` 命令，具体参数如下：

复制代码

```
sudo ip link set veth1 xdpgeneric off
```

除了 `ip link`之外， BCC 也提供了方便的库函数，让我们可以在同一个程序中管理 XDP 程序的生命周期：

复制代码

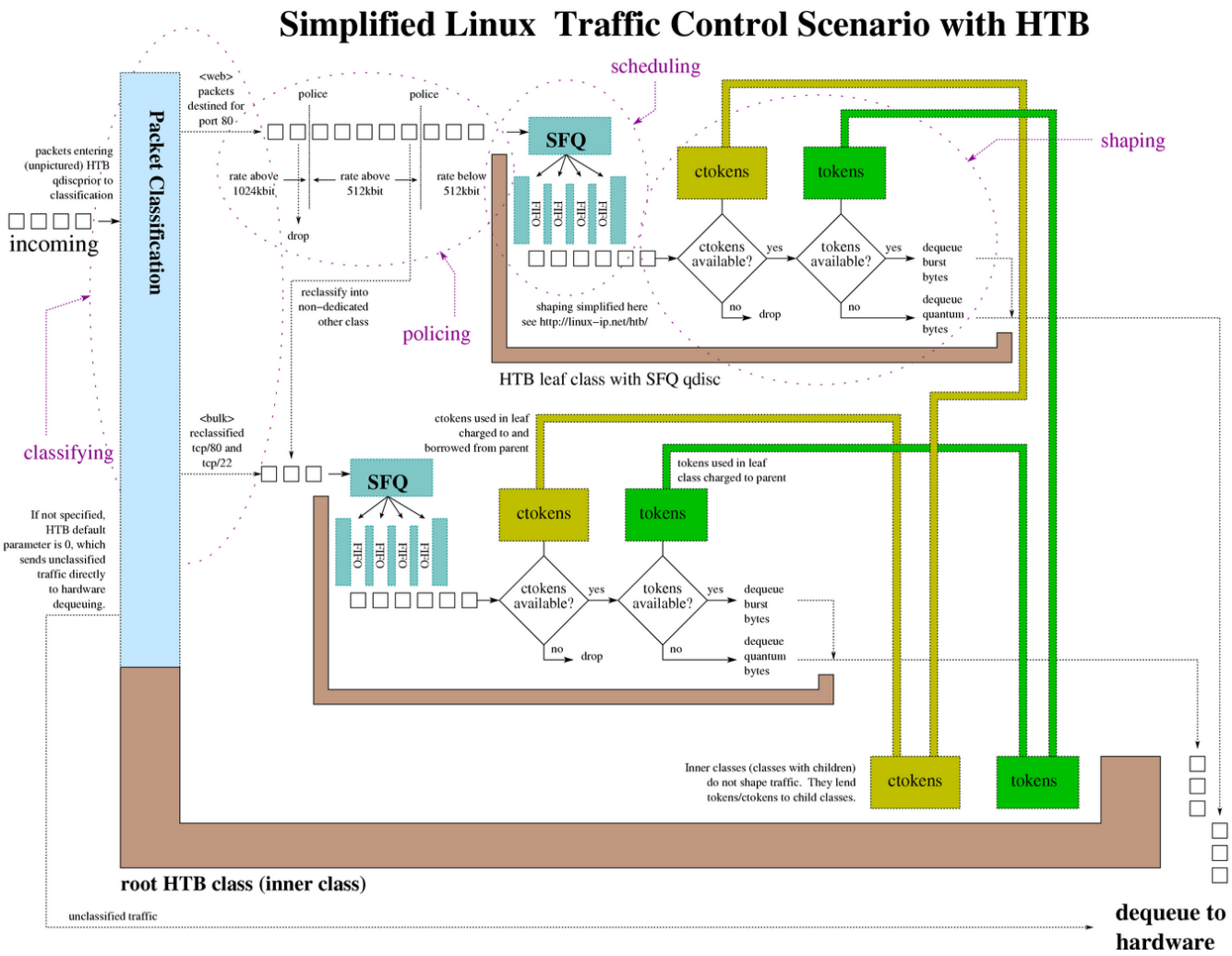
```
from bcc import BPF

# 编译XDP程序
```

```
b = BPF(src_file="xdp-example.c")
fn = b.load_func("xdp-example", BPF.XDP)

# 加载XDP程序到eth0网卡
device = "eth0"
b.attach_xdp(device, fn, 0)

# 其他处理逻辑
...

# 卸载XDP程序
b.remove_xdp(device)
```

TC 程序

TC 程序的类型定义为 BPF_PROG_TYPE_SCHED_CLS 和 BPF_PROG_TYPE_SCHED_ACT，分别作为 Linux 流量控制 的分类器和执行器。Linux 流量控制通过网卡队列、排队规则、分类器、过滤器以及执行器等，实现了对网络流量的整形调度和带宽控制。

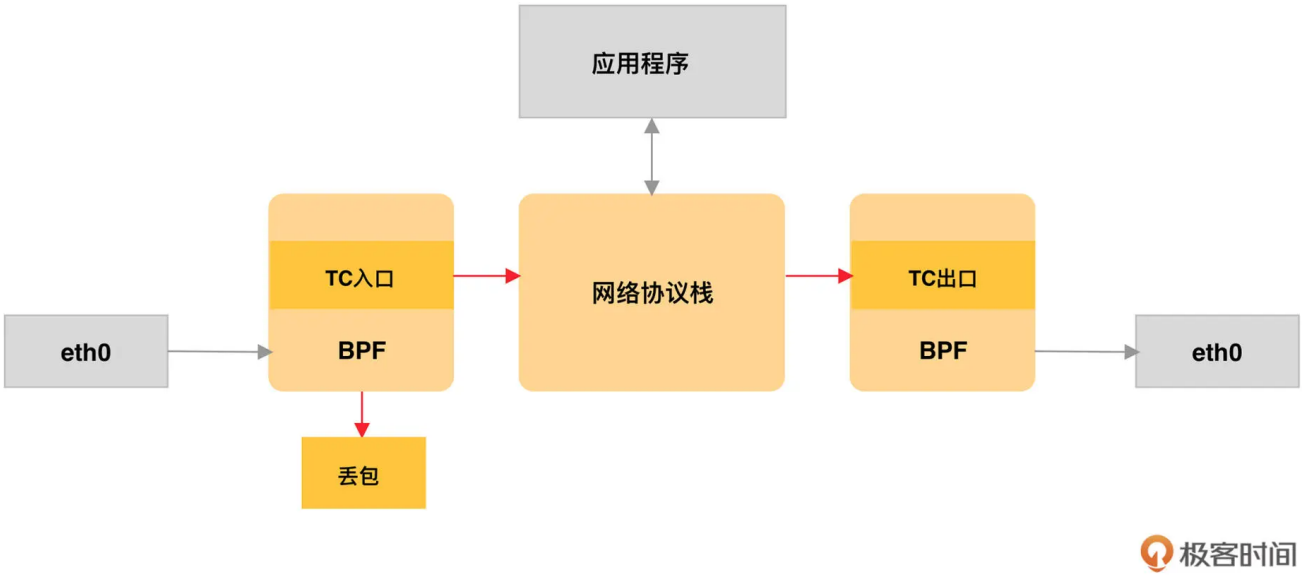
下图（图片来自 linux-ip.net）展示了 HTB（Hierarchical Token Bucket，层级令牌桶）流量控制的工作原理：



HTB 流量控制

由于 Linux 流量控制并非本课程的重点，这里我就不过多展开了。如果你对它还不熟悉，可以参考 官方文档 进行学习。

得益于内核 v4.4 引入的 direct-action 模式，TC 程序可以直接在一个程序内完成分类和执行的动作，而无需再调用其他的 TC 排队规则和分类器，具体如下图所示：



TC eBPF 程序与网络协议栈的关系

同 XDP 程序相比，TC 程序可以**直接获取内核解析后的网络报文数据结构**`sk_buff`（XDP 则是 `xdp_buff`），并且可**在网卡的接收和发送两个方向上执行**（XDP 则只能用于接收）。下面我们来具体看看 TC 程序的执行位置：

> 对于接收的网络包，TC 程序在网卡接收（GRO）之后、协议栈处理（包括 IP 层处理和 iptables 等）之前执行；
>
> 对于发送的网络包，TC 程序在协议栈处理（包括 IP 层处理和 iptables 等）之后、数据包发送到网卡队列（GSO）之前执行。

除此之外，由于 TC 运行在内核协议栈中，不需要网卡驱动程序做任何改动，因而可以挂载到任意类型的网卡设备（包括容器等使用的虚拟网卡）上。

同 XDP 程序一样，TC eBPF 程序也可以通过 Linux 命令行工具来加载到网卡上，不过相应的工具要换成 `tc`。你可以通过下面的命令，分别加载接收和发送方向的 eBPF 程序：

复制代码

```
# 创建 clsact 类型的排队规则
sudo tc qdisc add dev eth0 clsact

# 加载接收方向的 eBPF 程序
sudo tc filter add dev eth0 ingress bpf da obj tc-example.o sec ingress

# 加载发送方向的 eBPF 程序
sudo tc filter add dev eth0 egress bpf da obj tc-example.o sec egress
```

套接字程序

套接字程序用于过滤、观测或重定向套接字网络包，具体的种类也比较丰富。根据类型的不同，套接字 eBPF 程序可以挂载到套接字（socket）、控制组（cgroup ）以及网络命名空间（netns）等各个位置。你可以根据具体的应用场景，选择一个或组合多个类型的 eBPF 程序，去控制套接字的网络包收发过程。

这里，我把常见的套接字程序类型，以及它们的应用场景和挂载方法整理成了一个表格，你可以在需要时参考：

套接字程序类型	应用场景	挂载方法
BPF_PROG_TYPE_SOCKET_FILTER	用于套接字过滤和观测	用户态程序可通过系统调用 setsockopt(sock, SOL_SOCKET, SO_ATTACH_BPF, ...)，绑定 BPF 程序到具体的 socket 上
BPF_PROG_TYPE_SOCK_OPS	用于套接字修改或重定向	用户态程序可通过 BPF 系统调用的 BPF_PROG_ATTACH 命令（指定挂载类型为 BPF_CGROUP_SOCK_OPS），将其挂载到 cgroup 上
BPF_PROG_TYPE_SK_SKB	用于套接字修改或 消息流动态解析	用户态程序可通过 BPF 系统调用的 BPF_PROG_ATTACH 命令（指定挂载类型为 BPF_SK_SKB_STREAM_VERDICT 或 BPF_SK_SKB_STREAM_PARSER），将其挂载到 BPF_MAP_TYPE_SOCKMAP 类型的 BPF 映射上
BPF_PROG_TYPE_SK_MSG	用于控制内核是否发送消息到套接字	用户态程序可通过 BPF 系统调用的 BPF_PROG_ATTACH 命令（指定挂载类型为 BPF_SK_MSG_VERDICT）将其挂载到 BPF_MAP_TYPE_SOCKMAP 类型的 BPF 映射上
BPF_PROG_TYPE_SK_REUSEPORT	用于控制端口是否重用	用户态程序可通过系统调用 setsockopt(sock, SOL_SOCKET, SO_ATTACH_REUSEPORT_EBPF, ...)，绑定 BPF 程序到具体的 socket 上
BPF_PROG_TYPE_SK_LOOKUP	用于为新的 TCP 连接选择监听套接字，或为 UDP 数据包选择未连接的套接字，可用来绕过 bind 系统调用的限制	用户态程序可通过系统调用 bpf(BPF_LINK_CREATE, ...)，绑定 BPF 程序到网络命名空间（netns）上

极客时间

cgroup 程序

cgroup 程序用于**对 cgroup 内所有进程的网络过滤、套接字选项以及转发等进行动态控制**，它最典型的应用场景是对容器中运行的多个进程进行网络控制。

cgroup 程序的种类比较丰富，我也帮你整理了一个表格，方便你在需要时查询：

cgroup 程序类型	应用场景
BPF_PROG_TYPE_CGROUP_SKB	在入口和出口过滤数据包，并可以接受或拒绝数据包
BPF_PROG_TYPE_CGROUP_SOCK	在套接字创建、释放和绑定地址时，接受或拒绝操作，也可用来统计套接字信息
BPF_PROG_TYPE_CGROUP_SOCKOPT	在 setsockopt 和 getsockopt 操作中修改套接字选项
BPF_PROG_TYPE_CGROUP_SOCK_ADDR	在 connect、bind、sendto 和 recvmsg 操作中，修改 IP 地址和端口
BPF_PROG_TYPE_CGROUP_DEVICE	对设备文件的访问进行过滤
BPF_PROG_TYPE_CGROUP_SYSCTL	对 sysctl 的访问进行过滤

极客时间

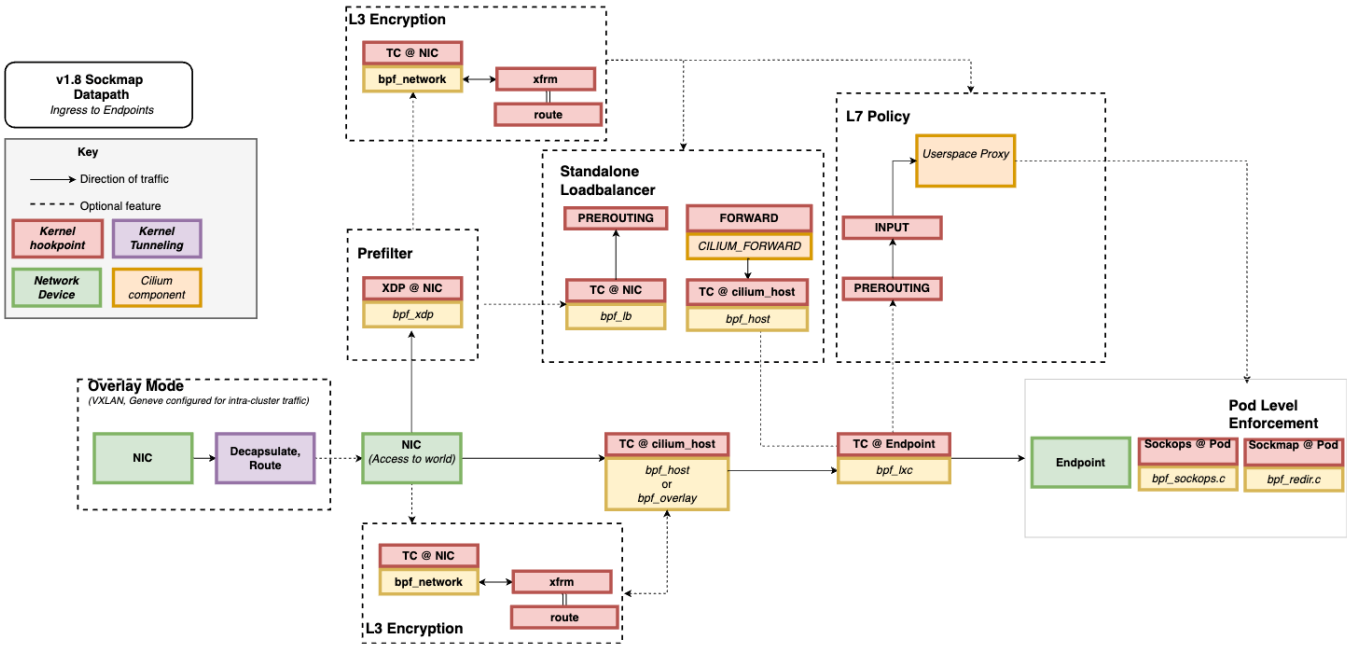
这些类型的 BPF 程序都可以通过 BPF 系统调用的 BPF_PROG_ATTACH 命令来进行挂载，并设置挂载类型为匹配的 BPF_CGROUP_xxx 类型。比如，在挂载 BPF_PROG_TYPE_CGROUP_DEVICE 类型的 BPF 程序时，需要设置 bpf_attach_type 为 BPF_CGROUP_DEVICE：

复制代码

```
union bpf_attr attr = {};
attr.target_fd = target_fd;            // cgroup文件描述符
attr.attach_bpf_fd = prog_fd;          // BPF程序文件描述符
attr.attach_type = BPF_CGROUP_DEVICE;  // 挂载类型为BPF_CGROUP_DEVICE

if (bpf(BPF_PROG_ATTACH, &attr, sizeof(attr)) < 0) {
  return -errno;
}

...
```

注意，这几类网络 eBPF 程序是在不同的事件触发时执行的，因此，在实际应用中我们通常可以把多个类型的 eBPF 程序结合起来，一起使用，来实现复杂的网络控制功能。比如，最流行的 Kubernetes 网络方案 Cilium 就大量使用了 XDP、TC 和套接字 eBPF 程

序，如下图（图片来自 Cilium 官方文档，图中黄色部分即为 Cilium eBPF 程序）所示：



Cilium eBPF 数据面

其他类 eBPF 程序

除了上面的跟踪和网络 eBPF 程序之外，Linux 内核还支持很多其他的类型。这些类型的 eBPF 程序虽然不太常用，但在需要的时候也可以帮你解决很多特定的问题。

我将这些无法划分到网络和跟踪的 eBPF 程序都归为其他类，并帮你整理了一个表格：

BPF 程序类型	应用场景
BPF_PROG_TYPE_LSM	用于 Linux 安全模块（Linux Security Module, LSM）访问控制和审计策略
BPF_PROG_TYPE_LWT_IN BPF_PROG_TYPE_LWT_OUT BPF_PROG_TYPE_LWT_XMIT	用于轻量级隧道（如 vxlan、mpls 等）的封装或解封装
BPF_PROG_TYPE_LIRC_MODE2	用于红外设备的远程遥控
BPF_PROG_TYPE_STRUCT_OPS	用于修改内核结构体，目前仅支持拥塞控制算法 tcp_congestion_ops
BPF_PROG_TYPE_FLOW_DISSECTOR	用于内核流量解析器（Flow Dissector）
BPF_PROG_TYPE_EXT	用于扩展BPF程序

极客时间

这个表格列出了一些不太常用的 eBPF 程序类型，你可以先大致浏览下，在需要的时候再去深入了解。

小结

今天，我带你一起梳理了 eBPF 程序的主要类型，以及不同类型 eBPF 程序的应用场景。

根据具体功能和应用场景的不同，我们可以把 eBPF 程序分为跟踪、网络和其他三类：

跟踪类 eBPF 程序主要用于从系统中提取跟踪信息，进而为监控、排错、性能优化等提供数据支撑；

网络类 eBPF 程序主要用于对网络数据包进行过滤和处理，进而实现网络的观测、过滤、流量控制以及性能优化等；

其他类则包含了跟踪和网络之外的其他 eBPF 程序类型，如安全控制、BPF 扩展等。

虽然每个 eBPF 程序都有特定的类型和触发事件，但这并不意味着它们都是完全独立的。通过 BPF 映射提供的状态共享机制，各种不同类型的 eBPF 程序完全可以相互配合，不仅

可以绕过单个 eBPF 程序指令数量的限制，还可以实现更为复杂的控制逻辑。

思考题

最后，我想邀请你来聊一聊：

1. 你是怎么理解 eBPF 程序类型的呢？
2. 如果让你来重新设计类似于 Cilium 的网络方案，你会如何选择 eBPF 程序类型呢？

期待你在留言区和我讨论，也欢迎把这节课分享给你的同事、朋友。让我们一起在实战中演练，在交流中进步。

分享给需要的人，Ta订阅超级会员，你将得 50 元

Ta单独购买本课程，你将得 20 元

生成海报并分享

赞 3 提建议

© 版权归极客邦科技所有，未经许可不得传播售卖。 页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 05 | 编程接口：eBPF 程序是怎么跟内核进行交互的？

下一篇 07 | 内核跟踪（上）：如何查询内核中的跟踪点？

精选留言 (3)

写留言

YF

2022-01-30

你是怎么理解 eBPF 程序类型的呢？

eBPF 对应与内核的事件类型，犹如订阅同类消息事件，内核发现对应的事件，则通知订阅者处理。

作者回复: 非常棒！

1

不了峰

2022-01-28

打卡

作者回复: 加油！

1

阿立

2022-02-10

作者大佬，xdp是一个内核网络处理模块，还是网络包进入协议栈之前的事件钩子点呢？ tc呢？

作者回复: 都是网络包事件钩子，当然也是网络处理流程中的一部分。XDP和TC其实有些类似，但TC总是在内核中执行，而XDP可以卸载到网卡硬件，从而获得更好的性能。